

# Data structures

Videogames Technology  
Escuela Politécnica Superior

Departamento de Automática

## Objectives

1. Understand the need to store information in data structures.
2. Identify most appropriate data structure according to the problem.
3. Understand the role of lists in Arcade.
4. Basic usage of sprites in Arcade.

# Table of Contents

## I. Data structures

- Introduction
- Array
- Stack and queue
- lists and hash tables
- Trees
- Graphs

## 2. Data structures in Arcade

- Sprites
- Sprites in Arcade

# Data structures

## Introduction

Programming is about information representation.

- Simple data are easy to represent: Numbers, characters, strings, etc.

Reality uses to be more complicated.

- A class represent an object.
- How can we store several objects?
- How can we represent complex data?

We need powerful mechanisms to store information: Data structures.

# Data structures

## Array

### Vector (1-D array)

0	1	2	3
$a_0$	$a_1$	$a_2$	$a_3$

### Matrix (2-D array)

	0	1	2	3
0	$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
1	$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
2	$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$

### Advantajes:

- Very fast

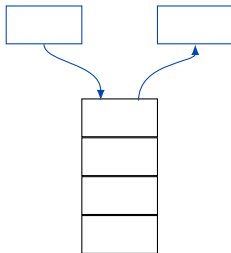
### Disadvantajes:

- Fixed size
- Not supported in Python by default
  - NumPy

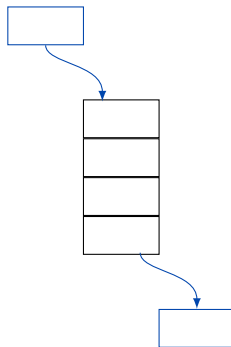
# Data structures

## Data structures (I): Stack and queue

Stack (LIFO)



Queue (FIFO)



Operations:

- `push(value)` and `pop(value)`

Implemented as lists in Python

# Data structures

## Lists and hash tables

Lists



Operations:

- `insert(pos, value)`
- `get(pos)`

Hash table

(associative array, dictionary)

Key 1	Value 1
Key 2	Value 2
Key 3	Value 3
Key 4	Value 4

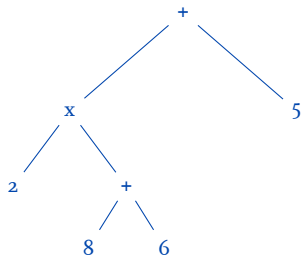
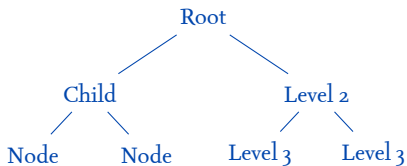
Operations:

- `put(key, value)`
- `get(key)`

# Data structures

## Trees (I)

### Trees



Operations:

- `insert()` and `remove()`
- `search()`

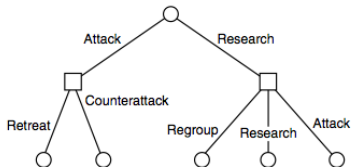
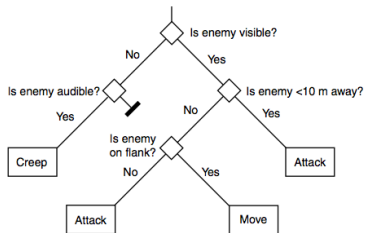
$$2 * (8 + 6) + 5$$



# Data structures

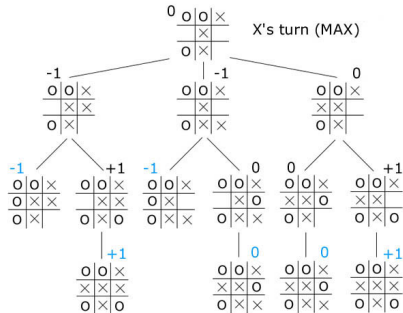
## Trees (II)

### Decision trees



Source: Ian Millington, John Funge. "Artificial Intelligence for Games". Ed. Morgan-Kaufmann. 2009.

### Search trees

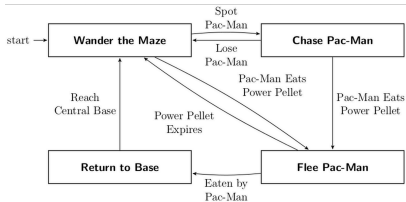
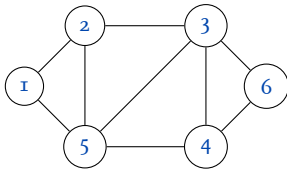


(Source)

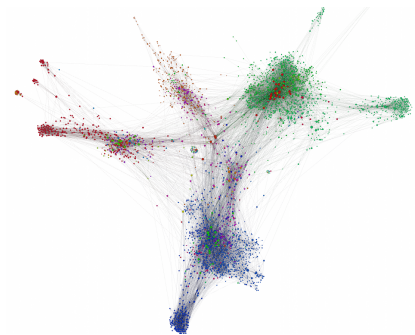
# Data structures

## Graphs

### Graphs



(Source)



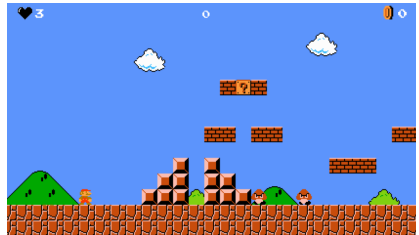
(Video Path-Planning)

# Data structures in Arcade

## Sprites (I)

### Sprite

A sprite is a 2D image used in videogames



# Data structures in Arcade

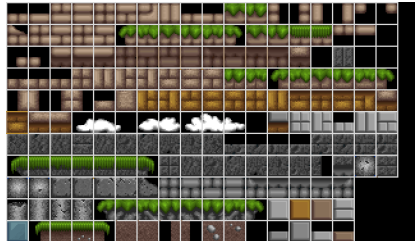
## Sprites (II)

A videogame contains many sprites

- Difficult maintenance
- Solution: Spritesheets

### Advantages

- One file contains many sprites
- Less I/O operations  $\Rightarrow$  Better performance
- Less memory consumption



# Data structures in Arcade

## Sprites (III)

In general, any data can be stored in three forms

- Not compressed
- Compressed with loss
- Compressed without loss

	Image format	Sound format	Binary data
Not compressed	BMP	WAV	
Compressed with loss	JPG	MP3	
Compressed without loss	PNG, GIF	-	ZIP, bzip, rar, ...

# Data structures in Arcade

## Sprites (IV)

Attending to what information is stored in image format, there are two types of image formats:

- Bitmap: stores each pixel
  - Scales bad
  - Formats: JPG, PNG, BMP, GIF
- Vectorial: stores coordinates
  - Scales well
  - Not supported by Arcade
  - Formats: SVG, EPS

Many open assets for your games!

- (Kenney)

# Data structures in Arcade

## Sprites in Arcade (I)

You will need to provide a **path** to the file

- **Absolute path:** Starts from the root directory
  - Example (Windows):  
`c:\\Users\\atreides\\Desktop\\mygame\\assets\\sprites\\mario.png`
  - Example (Linux):  
`/home/atreides/mygame/assets/sprites/mario.png`
- **Relative path:** Relative to the project's directory
  - Example (Windows): `assets\\sprites\\mario.png`
  - Example (linux): `assets/sprites/mario.png`

**Always** use relative paths in your projects!!!

# Data structures in Arcade

## Sprites in Arcade (II)

Sprites are a fundamental concept in Arcade

### Creating a sprite

```
character = arcade.Sprite('images/character.png')
```

### Placing a sprite

```
character.center_x = 300  
character.center_y = 200
```



# Data structures in Arcade

## Sprites in Arcade (II)

Arcade stores sprites in lists

```
wall = arcade.Sprite('images/boxCrate.png')  
wall.center_x = 300  
wall.center_y = 300  
wall_list = arcade.SpriteList()  
wall_list.append(wall)
```

Lists are manipulated as a whole

```
wall_list.draw()
```

And sprites can be removed from the list

```
wall.remove_from_sprite_lists()
```

# Data structures in Arcade

## Sprites in Arcade (III)

Lists in Arcade implements **collision detection** and handling

```
hit_list =  
arcade.check_for_collision_with_list(player_sprite,  
coin_list)
```

# Data structures in Arcade

## Sprites in Arcade (IV)

Functional example in (example)

# Data structures in Arcade

## Sprites in Arcade (VI)

Locating sprites in the game is a tough work

- Closely related to **level design**
- There are tools that ease this task

(Tiled Map Editor)